Lab11: Geolocation with Python

Overview

This lab extends the network traffic analysis from lab 9 by showing the IP addresses on a map. We also explore EXIF information in jpg pictures, with a particular focus on the geolocation information.

As we are coming to the end of the module, this lab is more exploratory than earlier ones – the emphasis is on understanding given code examples and then applying concepts from these examples yourself to different situations. This will require you to research selected concepts yourself.

All files you need for this lab can be downloaded from moodle as a single zip archive.

Part A - location of IP addresses

1. Check/install geoip2 and a geolocation database

The Python module **geoip2** was briefly introduced in the lecture. This and the next exercise is designed for you to explore this more.

In Python, try to import geoip2.database. If this fails install geoip2 using pip install (from the windows command line - running as administrator).

For geoip2 to work, it also needs a suitable geolocation database.

Download the Creative Commons-licenced **GeoLite2 City database** from MaxMind from https://dev.maxmind.com/geoip/geoip2/geolite2/ (Choose MaxMind DB binary format, Geolite2 City). Unzip the archive (I had to do it twice) until you have the actual database file called **GeoLite2-City.mmdb**. Copy/move this to a suitable directory, e.g. your Python37 directory.

Also note the License information included in the unzipped folder.

Note the name of the database file. My code below assumes that the filename is **Geo.mmdb**.

Name of your database file (with path):

Geoip2 documentation can be found at https://geoip2.readthedocs.io/en/latest/. Refer to the relevant parts of the documentation as you go along.

2. A first look at geoip2

Now you should be able to run the following code in IDLE without error. (If your database is in a different directory or has a different name, you will need to change the path accordingly):

```
import geoip2.database
reader = geoip2.database.Reader(r'C:\Program Files\Python37\Geo.mmdb')
type(reader)
rec = reader.city('146.176.164.91')
print(rec)
```

Q: What data type is the reader object?

Q: What data type is rec? What does this seem to contain?

Q: What is the name of the key that contains the country code in 2 letter ISO format, like 'GB'?

Q: What are the names of the key and subkeys that contain the longitude and latitude?

Q: What is the location in (longitude, latitude) format? How many decimal places?

Q: Which key contains the postcode? Does it contain the full postcode?

Q: Which keys contain the city name and the country name? Is there more than one name for each? Why? How do you get the city and country names in French?

Q: Why should the reader object only be created once?
(refer to https://geoip2.readthedocs.io/en/latest/)?

Test what you found out by executing the following code:

```
rec.country.iso_code
rec.location.longitude
rec.location.latitude
rec.postal.code
rec.country.name
rec.city.name
rec.country.names['fr']
rec.city.names['fr']
```

Now execute:

```
rec2 = reader.city('147.176.164.91')
print(rec2)
```

Q: What are the city and postcode of this IP address?

Q: Which country is this IP address in? What is the location in (longitude, latitude) format?

Q: Do you notice a difference in the accuracy of the geolocation information?

Q: Is the information for this IP address complete?

Now execute:

```
        rec3 = reader.city('192.168.255.255')
```

Q: What is the result?

Q: What data type is rec3?

Q: Can you explain this? (Hint – have you seen this address before, in other modules? Where?)

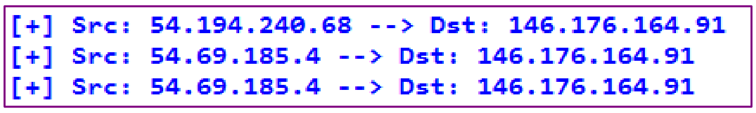
So, we can conclude that reader.city() returns geolocation information about the given IP address. This is in the form of a geoip2.models.City object, which in turn contains several dictionaries.

If the IP address is not in the database (e.g. if it is a private address), reader.city() returns an error.

3.   Add city and country to pcap analysis with geoip2

(See lecture slides)

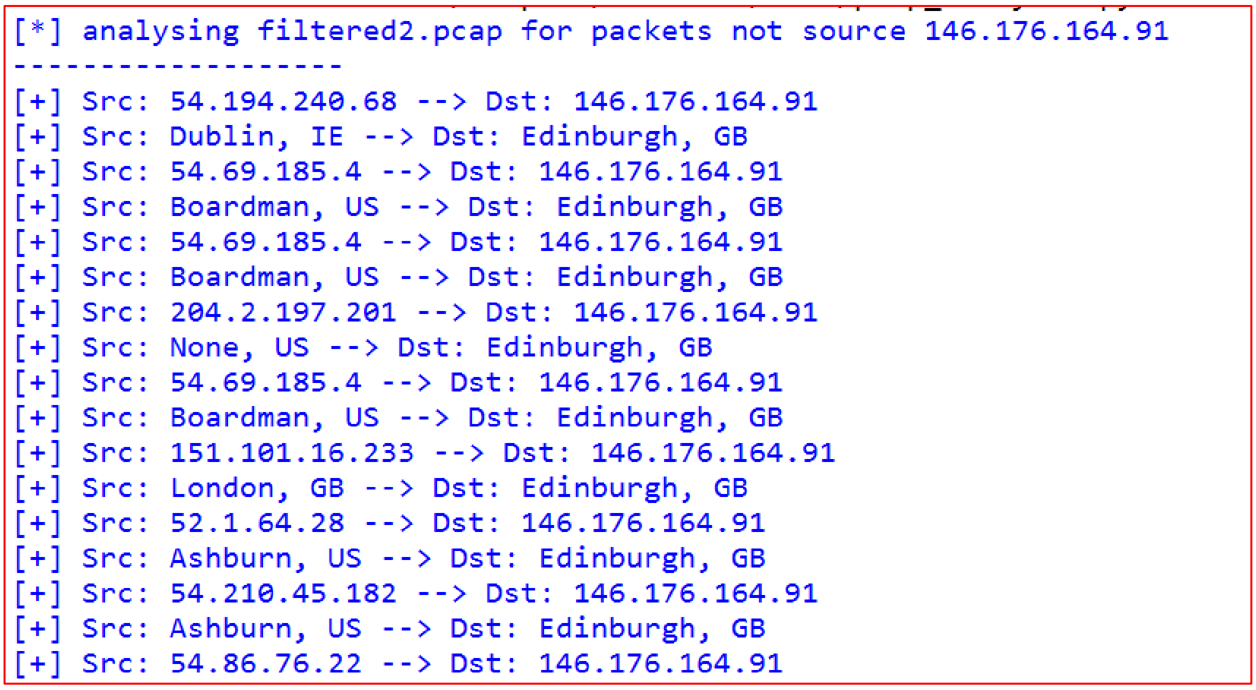
In lab 9, you were given a script pcap_downloads.py. You were then asked to use methods from the script to write a new script **pcap_exclude.py** that will show traffic source and destination for each packet from a pcap wireshark network capture in the format:



```
[+] Src: 54.194.240.68 --> Dst: 146.176.164.91
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
```

In case you didn't get this finished yourself, a working script that achieves this is now provided for you – **pcap_exclude.py** in moodle.

Using **pcap_exclude.py** as the starting point, create a new script **pcap_analysis.py** which also shows the city and 2-letter country code for each of the addresses, in the format:



```
[*] analysing filtered2.pcap for packets not source 146.176.164.91
-------------------
[+] Src: 54.194.240.68 --> Dst: 146.176.164.91
[+] Src: Dublin, IE --> Dst: Edinburgh, GB
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: Boardman, US --> Dst: Edinburgh, GB
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: Boardman, US --> Dst: Edinburgh, GB
[+] Src: 204.2.197.201 --> Dst: 146.176.164.91
[+] Src: None, US --> Dst: Edinburgh, GB
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: Boardman, US --> Dst: Edinburgh, GB
[+] Src: 151.101.16.233 --> Dst: 146.176.164.91
[+] Src: London, GB --> Dst: Edinburgh, GB
[+] Src: 52.1.64.28 --> Dst: 146.176.164.91
[+] Src: Ashburn, US --> Dst: Edinburgh, GB
[+] Src: 54.210.45.182 --> Dst: 146.176.164.91
[+] Src: Ashburn, US --> Dst: Edinburgh, GB
[+] Src: 54.86.76.22 --> Dst: 146.176.164.91
```

Hints:

- Write a function to look up city and iso_code for a given IP address. Once the function is working, add it to the script.
- Add a print statement to the existing function printPcap in the script which prints a second row for each packet underneath the IP addresses, showing city and country code for the source and the destination.
- Test your script with filtered2.pcap and filtered3.pcap (from lab 9).
- Refer to the lecture slides for further help.

Part B - KML, EXIF

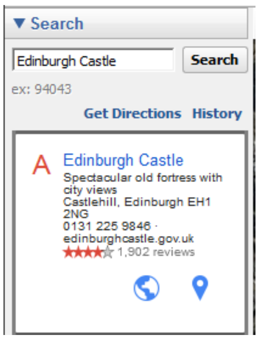
4. Explore Google Earth and KML

In the lecture, we introduced KML. Let's explore this in practice.

Open Google Earth.

You will see the search box in the top left. Search for Edinburgh Castle.

The satellite map will fly you there, add a place marker, and display the corresponding record →



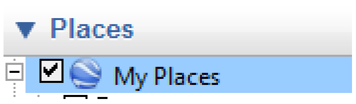
Edinburgh Castle

ex: 94043

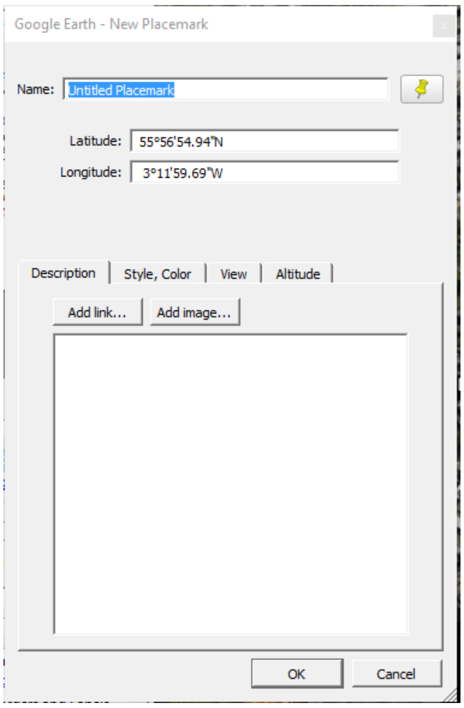
A Edinburgh Castle
Spectacular old fortress with city views
Castlehill, Edinburgh EH1 2NG
0131 225 9846 ·
edinburghcastle.gov.uk
★★★★☆ 1,902 reviews

When you right-click the Edinburgh Castle link, you can choose copy as KML. However, this doesn't work for me – when I open a text editor – notepad, or notepad++, I can't paste. Try it yourself. If you have the same problem, try a different approach.

Right-click on My Places and choose Add > Placemark.



A new pushpin will flash on the map and the new placemark dialog will open →



Name: Untitled Placemark

Latitude: 55°56'54.94"N

Longitude: 3°11'59.69"W

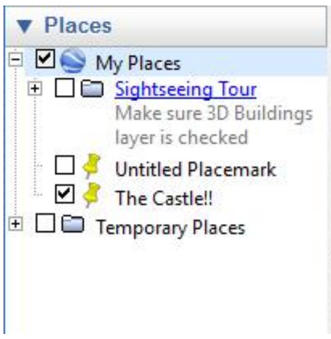
Move the pin on the map to Edinburgh Castle, change the name of the placemark to "The Castle!!" and click ok.

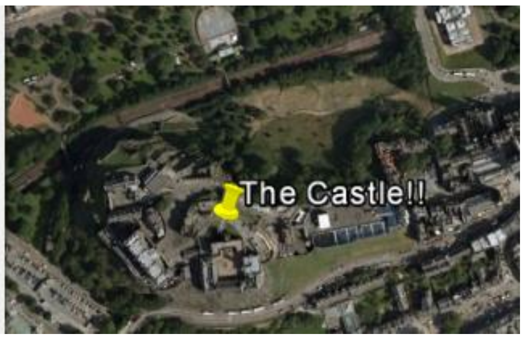
Close the Search box

and

Un-tick all layers from the Primary Database,

so that you can view your new Placemark on its own.





Right-click on "The Castle!!" in My Places and select Copy.

Open notepad or Notepad++ and paste.

Have a good look at the KML that you have created. While this file is far more complex than is actually required, you should be able to identify the XML header
<?xml version="1.0" encoding="UTF-8"?> and the <kml> root element.

The <Placemark> element contains the information about our new marker. <LookAt> is not a required element within Placemark, but <Point> is.

The longitude and latitude are recorded in the <coordinates> element within the <Point> element. There is a third value – probably 0 (the default) – which indicates the altitude.

5. Create your first KML file with Python

Now that you understand KML and google Earth, let's create our first KML file with Python. Open **napier_KML.py** (from moodle) in the Python IDLE:

```
# napier_KML.py
# simple example to show creation of KML file with one point
# Petra 13/11/16
# last updated Nov 2018  (PEP8)

import simplekml

# define KML object
kml = simplekml.Kml()
# add a single Point
pnt = kml.newpoint(name="ENU",
                   coords=[(-3.2136, 55.932892)],
                   description="Merchiston Campus")
kml.save("napier.kml")  # save
print(kml.kml())  # print kml to screen
```

Run the script.

Q: Do you get an error? If yes, the problem is probably that you haven't installed simplekml yet so it can't be imported. Install it using pip install and run the script again. It should work now!

Q: Look at the screen output, which is a copy of the KML that was written to file. Can you spot the name, coordinates and description in the Placemark?

Q: Look at the directory where your script is. Does this contain a new file called napier.kml?

Q: How could you have changed the script to place the kml file in a different directory?

Open Google Earth. Click File > Open and open napier.kml.

Q: Where does your new Placemark appear in My Places?

Q: Is your marker for the Castle still there? (Note that you can check/ uncheck the places in My Places that you do / don't want to see)

6. Geolocation of photos from EXIF information

In the lecture, I showed you a mini project that extracts geolocation from photos and plots this on a static map.

The script **exif.py** which I used for this is included in the zip in moodle, along with some example photos.

Re-read the relevant slides from the lecture. Then explore this script and play around with it. The script should run as long as you change the folder paths etc appropriately, and remember to pip install any modules that won't import initially.

For the map to open, you will also need to add a valid Google maps API key to the script where indicated. You can find such a key in moodle. The use of this key is currently restricted to Edinburgh Napier University IP addresses, so if you are off campus, you will need to use ENU VPN or virtual desktop, or create your own key by signing up for a free trial at http://g.co/dev/maps-no-account.

The easiest way to run the script with a different selection of pictures is to keep all the photos in a backup folder somewhere and then just copy the ones you want to use to the folder that is read by the script.

7. ***OPTIONAL*** Improving exif.py

The exif.py script is very rough and ready – I threw it together in a hurry for a demonstration for schoolkids. It was originally written for Python 2.7, and could be simplified considerably e.g. by using f-strings.

- It uses only one function and has no main() nor boiler-plate.
- It is not PEP8 compliant.
- It creates a static map, which only opens when a valid API key is provided, rather than writing a kml file which could be used more widely by any mapping apps, not just google earth.

Your task is to improve the script to address these shortcomings, and any others that you notice.

8. Putting it all together: IP addresses on a map

Now you are ready for today's main exercise.

Create a new script, **pcap_kml.py**, which is based on pcap_analysis.py.

The script should:

- Read a specified pcap file
- Extract the IP addresses of the **source** of each packet (initially to a list)
- Make a dictionary from the list of IP addresses, where the IP address is the key and the value is e.g. **5 packet(s)** [i.e. the value is the number of packets with the source IP given by the key].
  Hint: you can use collections.counter() for this.
- Write a KML file that contains a Placemark for each **source IP address** that has a known geolocation.
  The Description should show the packet count.

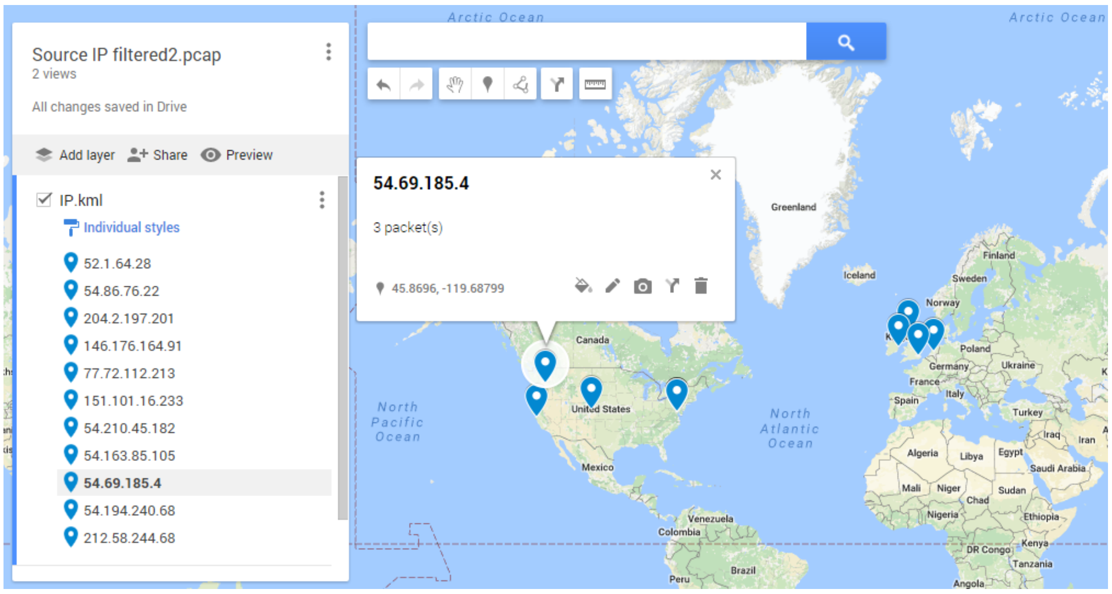
Remember to develop your script bit by bit (one bullet point at a time). Use print statements to check that each part of your script works. Once you are satisfied, move on to the next bullet point.
Once you have finished, run the script for filtered2.pcap and filtered3.pcap.
If you like, you could also change it very easily to use destinations rather than sources.
Remember to change the name of the kml file you're writing to each time if you don't want to overwrite your previous file.

Open Google Earth and Import your new kml file.
Explore the result.

Q: Why is Google Earth not great for viewing the locations of our source packets?

Google Earth doesn't work very well because the locations are scattered across Europe and North America, and therefore difficult to view together on a globe.

Go to https://www.google.com/maps/d/ (you may need to log into google) and import your kml file.
It should look better as a map – mine is shown below. To view my map in a browser, go to https://drive.google.com/open?id=1n75HUNPd888rw9de81Q9czb1kGY&usp=sharing.
This shows the source IPs from the file filtered2.pcap.


Source IP filtered2.pcap
2 views
All changes saved in Drive

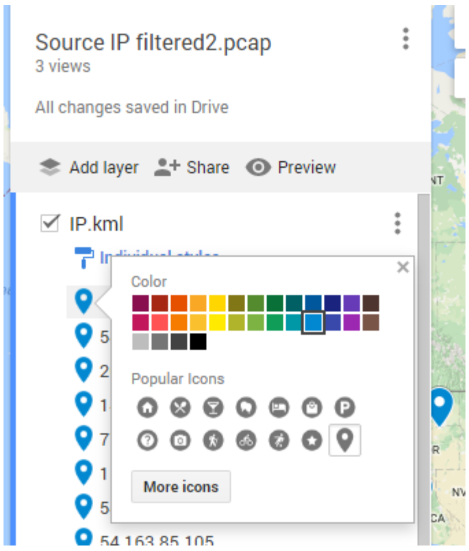
IP.kml
Individual styles
52.1.64.28
54.86.76.22
204.2.197.201
146.176.164.91
77.72.112.213
151.101.16.233
54.210.45.182
54.163.85.105
54.69.185.4
54.194.240.68
212.58.244.68

54.69.185.4
3 packet(s)
45.8696, -119.68799

9. ***Optional***: Final Enhancements

1. To improve your visualisation, go back to your python script and change it so that the description of each Placemark also includes the city and country.
   This should not be too hard if you apply ideas from the earlier pcap_analysis.py script, using geoip2 again to extract city and country.

2. It would be nice to change the icon, icon colour or even icon size to reflect the packet count.


Source IP filtered2.pcap
3 views
All changes saved in Drive

IP.kml

This can be done manually in Google My Maps by hovering over a point in the menu on the left and clicking on the little edit icon that will appear on its right, which brings up this menu →

Play about with this – change the style and colour of a few of the icons manually.
Once you are happy, click on the three vertical dots menu next to the map name (the one at the top) and choose Export to KML.
Export the IP.kml layer only. Save the file.

Now open the saved KML file in notepad++ and inspect it. You should find that the icon itself is determined by the <styleUrl> element, which can also be created through the functions included in the simplekml module. With a bit more research, you should be able to achieve this.